I have completed everything except A* search taking traffic lights and restrictions into account.

The start node is chosen with left click and the goal node is chosen with right click

A* Search algorithm sudocode:

```
Input: start node, goal node Output: Shortest path from start to goal
Set all nodes to unvisited
Initialise priority queue with <start, null, 0, start.estimatedCost(goal, costType)
//<node,parent,costFromStart, costFromStart+estimatedCostToGoal>, costType is either distance or
//time
While(queue is empty){
    Poll <node*, parent*, g*, f*>
    If(node* is unvisited){
        Set node* to visited
        If(node* == goal){break;}
        For(all neighbours of node* that are connected through outgoing edges){
            If(neighbour is not visited){
                Add <neighbour, node*, g(),f()> //where g() is cost from start and f()
is cost from start + estimated cost to goal
            }
        }
    }
}
```

Articulation Point Finding Algorithm Sudocode:

```
Initialise count and parent of all nodes to null, articulationPoints = {};
Set root to first node
Set root.count to 0
Set numberOfSubtrees to 0
For(each neighbour of root){
    If(neighbour.count == null){
        articulationPoints.addAll(iterativeArticulationPoints(neighbour, 0, root));
        visitedNodes.add(all nodes used in it iterativeArticulationPoints(neighbour, 0, root));
        numberOfSubtrees++;
        }
        If(numberOfSubtrees > 1){add root to articulationPoints
}
Set root to unvisited node and repeat until all nodes are visited
-----------------------------------------------------------------------------------
iterativeArticulationPoints(node, count, root){
        Initialise stack with <nodeWrapper(node, count, nodeWrapper(root, 0, null)>
        Initialise number to count;
        While(stack not empty){
                NodeWraper nodeWrapper = Peek <node*, count*, parent*>
                If(node*.count == null){
                        Node*.count = count*;
                        nodeWrapper.reachBack = count*;
                        set children of nodeWrapper to all but parent*
                }else if (nodeWrapper.child is not empty){
                        Get child from children and remove it
                        If(child.count != null){
                        nodeWrapperOfChild.reachBack = min(child.count,
                                nodeWrapper.reachBack);
                        }



                        else {
                                number++;
                                push<WrapperNode(child, number, nodeWrapper of parent to
                child)>;
                        }


                } else {
                        If(node* != node){
                                Node*.parent.reachBack = min(node*.reachBack, node*.
        parent.reachBack);
```

```
                    }
                    if(node*.reachBack >= node*. parent.count) {
                        articulationPoints.add(node. parent.node);
                    }
                    Remove node* from the stack and add it to visitedNodes


                }
            }
}
```

Distance:

When the cost that the A* Search is optimising is distance, the cost (or g()) is the length of the segments between nodes and the heuristic (f()-g()) is the length from the node to the goal node as the crow flies.

Time:

When the cost that the A* Search is optimising is time, the cost is the length of the node divided by the speed limit (t = d/v), and the heuristic is the length from the node to the goal node/110 as 110km/hr is the fastest speed you can go according to the data if we discount no speed limit. It may look like you change nothing by dividing the length by 110 but by doing this you ensure the heuristic is never larger than the actual value.

I tested my program by slowly going through the algorithms with the debugger and comparing them to the sudo-code/sudo-method and changed the code when there was a difference. I also made sure to test some edge cases, like only one segment between start and goal nodes, the same route forwards and backwards where there aren’t any one-way roads (and where there are).

I also printed out the number articulation points to work out if I had done it correctly.